####【置顶】资源汇集
如何自学Android - Android技术漫谈 - SegmentFault
Android学习之路
Android学习网站汇集
想写个 App 练手，有什么有趣的 API 接口推荐吗？ - 知乎
AndroidDevTools（sdk、ndk、androidstudio下载地址）
国内第三方推送平台：极光推送、个推、百度推送、小米推送

####【置顶】笔记
app-慕课网系列_笔记

####【置顶】一些功能封装及代码备忘录
android快速开发--常用utils类 - linghu_java的专栏 - 博客频道 - CSDN.NET
这里所有的commit操作使用了SharedPreferencesCompat.apply进行了替代，目的是尽可能的使用apply代替commit。因为commit方法是同步的，并且我们很多时候的commit操作都是UI线程中，毕竟是IO操作，尽可能异步；所以我们使用apply进行替代，apply异步的进行写入。
Android常用功能封装-Accumulating

####【置顶】Android_cmd

####工具
Smali、Dedexer、Dexdump、Dexjar、objdump等分析工具

Apktool：反编译/打包apk
dex2jar：apk文件的解压内容，会发现其中有一个classes.dex文件，该文件中存放的就是java代码了，此时需要用到dex2jar工具进行转换，得到jar。
jd-gui：软件打开后有个UI界面，选择刚刚转换得到的classes-dex2jar.jar文件，打开就是我们需要的java代码文件了
Systrace：其实有些类似Traceview，它是对整个系统进行分析（同一时间轴包含应用及SurfaceFlinger、WindowManagerService等模块、服务运行信息），不过这个工具需要设备内核支持trace（命令行检查/sys/kernel/debug/tracing）且设备是eng或userdebug版本才可以
Traceview(Android studio已集成)：记录了应用程序中每个函数的执行时间
HierarchyViewer：进行UI布局复杂程度及冗余
Lint：进行资源及冗余UI布局等优化
Leakcanary：可以感知泄露且定位泄露，实质是MAT原理，只是更加自动化了，无法很快察觉掌控全局代码时极力推荐；或者是偶现泄露的情况下极力推荐。
MAT：是一个专门分析Java堆数据内存引用的工具，我们可以使用它方便的定位内存泄露原因，核心任务就是找到GC ROOT位置即可。
bootchart：是一个用于linux启动过程性能分析的开源软件工具，在系统启动过程自动收集CPU占用率、进程等信息，并以图形方式显示分析结果，可用作指导优化系统启动过程。
Olly DBG, IDA使用
pc上的数据库工具：Navicat Premium
Bmob云存储
百度BAE

自动化测试相关的四个工具：
1、Monkeyrunner：优点：操作最为简单，可以录制测试脚本，可视化操作；缺点：主要生成坐标的自动化操作，移植性不强，功能最为局限；
2、Rubotium：主要针对某一个APK进行自动化测试，APK可以有源码，也可以没有源码，功能强大；缺点是针对APK操作，而且需要对APK重新签名（有工具），因此操作相对复杂；
3、UiAutomator：优点：可以对所有操作进行自动化，操作简单；缺点：Android版本需要高于4.0，无法根据控件ID操作，相对来说功能较为局限，但也够用了；

4、Monkey：准确来说，这不算是自动化测试，因为其只能产生随机的事件，无法按照既定的步骤操作；

==========hide==========

####待梳理

绘制图像的两种方法，canvas.drawBitmap(bitmap, 0, 0, null), drawable.draw(canvas);
java启动流程、openGL、videoview/surfaceview、单例、反射/代理、getApplication、widget、foreGround、进程间通信、开机向导、broadcastReceiver定义接收的先后顺序、app的启动过程、contentProvider机制、集合适用情况
include/merge/viewstub
testFragment2
有关reflect

####待复习

51.Android ContentProvider本地数据存储 - android安卓项目开发实战实例经典入门视频教程-极客学院

####有关面试

Android 面试要点
Android 面试题（答案最全） - Android面试题 - 职友集

####framework部分

1. DisplayManager
2. apk安装之devmapper方式
3. 静默安装、apkInstaller
4. 假待机， PowerManagerService假待机：action_screeen_off广播、关输出、屏蔽按键、灭灯； 真待机：断网络、灭灯、待机
5. PhoneWindowManager中键值截获及定制处理，比如启动定制UI的音量条

6. 定制数据库内容，在升级、恢复出厂设置后生效
7. 指定启动的launcher，防替换。ActivityManagerService 中startHomeActivityLocked替换掉homeIntent，PhoneWindowManager中截获KEYCODE_HOME。
8. 恢复出厂设置定向擦写数据
9. 保护重要app不被杀，ActivityManagerService中killUnneededProcessLocked中过滤。
10. intent、broadcast、service、handler、AIDL

==========hide==========

==========updating==========

####有关获取

####

android的wake_lock_郑传斌_新浪博客

Wake Lock是一种锁的机制, 只要有人拿着这个锁,系统就无法进入休眠，可以被用户态程序和内核获得. 这个锁可以是有超时的或者是没有超时的,超时的锁会在时间过去以后自动解锁. 如果没有锁了或者超时了, 内核就会启动休眠的那套机制来进入休眠.

/sys/power/wake_lock

android的PowerManager和PowerManager.WakeLock - DotDot - 博客园

1.各种锁的类型对CPU 、屏幕、键盘的影响：

PARTIAL_WAKE_LOCK:保持CPU 运转，屏幕和键盘灯有可能是关闭的。

SCREEN_DIM_WAKE_LOCK：保持CPU 运转，允许保持屏幕显示但有可能是灰的，允许关闭键盘灯

SCREEN_BRIGHT_WAKE_LOCK：保持CPU 运转，允许保持屏幕高亮显示，允许关闭键

盘灯

FULL_WAKE_LOCK：保持CPU 运转，保持屏幕高亮显示，键盘灯也保持亮度

2.另外WakeLock的设置是 Activiy 级别的，不是针对整个Application应用的。

解析activity之间数据传递方法的详解_Android_脚本之家

1.intent 2.public static成员变量 3.外部存储：SharePreference、SQLite、Content Provider和File 4.基于IPC的通信机制 5.基于Application Context.

Android 动画总结

Android 运行时请求权限 - CSDN博客

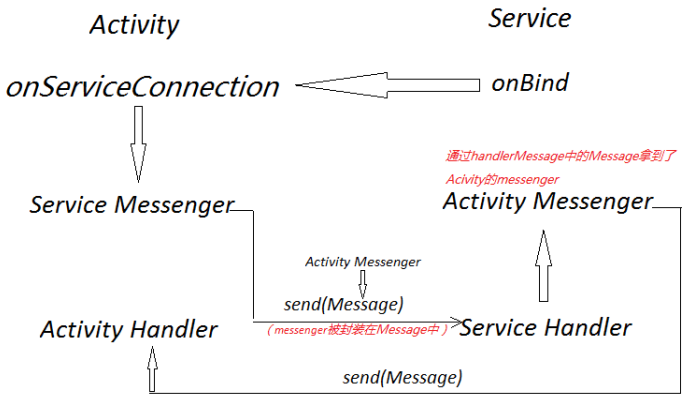
谈谈我的Android多渠道打包方式 - CSDN博客

Activity#onUserLeaveHint()和Activity#onUserInteraction() - 一口仨馍 - CSDN博客

Activity#onUserLeaveHint()用户手动离开当前activity，会调用该方法，比如用户主动切换任务，短按home进入桌面等。系统自动切换activity不会调用此方法，如来电，灭屏等。Activity#onUserInteraction() activity在分发各种事件的时候会调用该方法，注意：启动另一个activity,Activity#onUserInteraction()会被调用两次，一次是activity捕获到事件，另一次是调用Activity#onUserLeaveHint()之前会调用Activity#onUserInteraction()。

Android 生成系统签名的KeyStore - li411816761的专栏 - CSDN博客

使用Handler实现Activity和Service之间的交互



Android 禁止屏幕休眠和锁屏的方法 - Antonius的专栏 - 博客频道 - CSDN.NET

UnsatisfiedLinkError X.so is 64-bit instead of 32-bit之Android 64 bit SO加载机制 - 博客

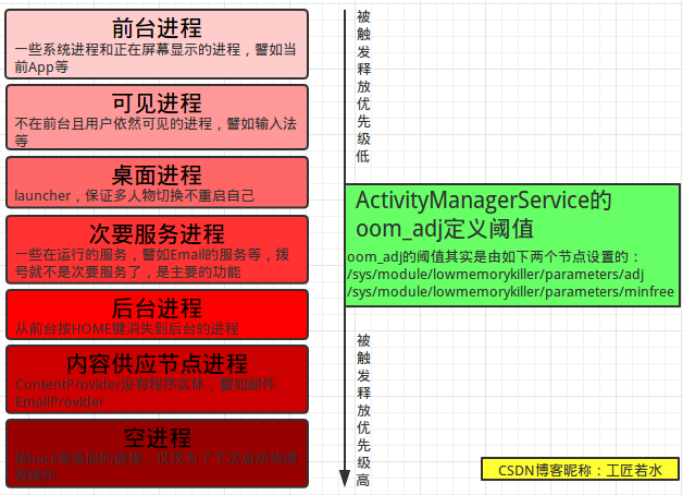
频道 - CSDN.NET

64-bit/32-bit 动态库不能混用

64-bit处理器可以向下兼容32-bit指令集，即可以运行32-bit动态库

张明云：Android应用内存泄露分析、改善经验总结

【mark】Android应用开发性能优化完全分析



【mark】Android 和 Java 性能优化最佳实践

NDK编译可执行文件在Android L中运行显示error: only position independent executables (PIE) are supported.失败问题解决办法。 - hxdanya的专栏 - 博客频道 - CSDN.NET

刘剑锷：探讨秋招各大顶级公司关于AsynTask的面试问题，求大神指点！！？

1.AsyncTask为什么要设计为只能够一次任务?

因为要同一个实例执行多次很麻烦，没必要。假设允许多次 execute 同一个 AsyncTask 多次，也就是说队列中会有同一个 AsyncTask 实例出现多次。首先 AsyncTask 需要改动很多，比如把各种状态额外保存多份，不能像现在这样简单做为 AsyncTask 的成员变量；如果 execute 时如果用了自己定义 Executor，是有可能会多线程同时访问的。

2.AsynTask造成的内存泄露的问题怎么解决，比如任务还在请求中，但这个Activity已经被销毁了，这个task就仍然持有act的引用，造成内存泄露?

最主要的是在 Activity 销毁时就应该把所有属于这个 Activity 的 Task cancel 掉。你的 Task 应该在 onCancelled 函数中做相应的处理。然后在 onCancelled 中将指向 Activity 的引用设为 null；弱引用更多的是一份保险，保证如果你在没有正确 cancel Task 时，不会让本应去死的 Activity 还因为 Task 的引用还在内存中晃悠。

3.Act销毁但Task如果没有销毁掉，当Act重启时这个task该如何解决？最多就是 Task 指向 Activity 的引用改成弱引用了。 Task 如果是 Activity 一个成员的话已经泄漏无法访问了。

Android BitmapShader 实战 实现圆形、圆角图片 - Hongyang - 博客频道 - CSDN.NET
Android自定义控件实战——水流波动效果的实现WaveView - vrix的专栏 - 博客频道 - CSDN.NET
Android照片墙完整版，完美结合LruCache和DiskLruCache - fangzhibin4712的专栏 - 博客频道 - CSDN.NET
安装Intel HAXM为Android 模拟器加速，30秒内启动完成 - 推酷
不使用Cygwin，在eclipse中快速开发JNI，一键生成C头文件.h，以及一键使用NDK交叉编译 - 开源中国社区
android使用软引用构建缓存 - hbzh2008的专栏 - 博客频道 - CSDN.NET
Java内部类与final关键字详解 - 塞外游侠的专栏 - 博客频道 - CSDN.NET
方法内部类 不能有访问修饰符，比如public
被匿名内部类访问的局部变量必须被final修饰
静态内部类不能访问外部类的非静态成员
无论内部类是公开的还是私有的，都不会被继承，因为他不是属性，也不是方法。而是一个内部事务的描述，我们称之为内部类
对于成员内部类来说，他会持有一份外部类当前对象的引用，Outer.this
方法内部类访问的局部变量必须被final修饰（为了约束两个不同变量的一致性。根本原因是内部类对象无法访问局部变量，才会去复制一份。
为了保证两个变量的一致性，才去使用final关键字修饰局部变量。而不是因为栈生命周期与堆生命周期不一致的问题）
在Java中，方法的局部变量位于栈上，对象位于堆上。
ActivityGroup相关--getLocalActivityManager() 以及intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)用法 - getchance的专栏 - 博客频道 - CSDN.NET
Android ListView使用BaseAdapter与ListView的优化 - OPEN 开发经验库
Android中使用ViewPager实现应用使用透明引导 - android5k - 博客园
Android下拉刷新完全解析，教你如何一分钟实现下拉刷新功能 - 郭霖的专栏 - 博客频道 - CSDN.NET
java.lang.ClassNotFoundException: Didn't find class "*****Activity" on path:

/data/app/*******.apk - lovexieyuan520的专栏 - 博客频道 - CSDN.NET
Android的硬件加速及可能导致的问题 - Ming's Blog
android获取多媒体文件的缩略图
解决Fail to create the java Virtual Machine_百度经验
Android 开发 之 JNI入门 - NDK从入门到精通 - 推酷

JNIEnv作用 : JNIEnv 是一个指针, 指向了一组JNI函数, 这些函数可以在jni.h中查询到, 通过这些函数可以实现 Java层 与 JNI层的交互, 通过JNIEnv 调用JNI函数 可以访问java虚拟机, 操作java对象;

JNI线程相关性: JNIEnv只在当前的线程有效,JNIEnv不能跨线程传递, 相同的Java线程调用本地方法, 所使用的JNIEnv是相同的, 一个Native方法不能被不同的Java线程调用;

JNIEnv结构体系: JNIEnv指针指向一个线程相关的结构,线程相关结构指向一个指针数组,指针数组中的每个元素最终指向一个JNI函数.

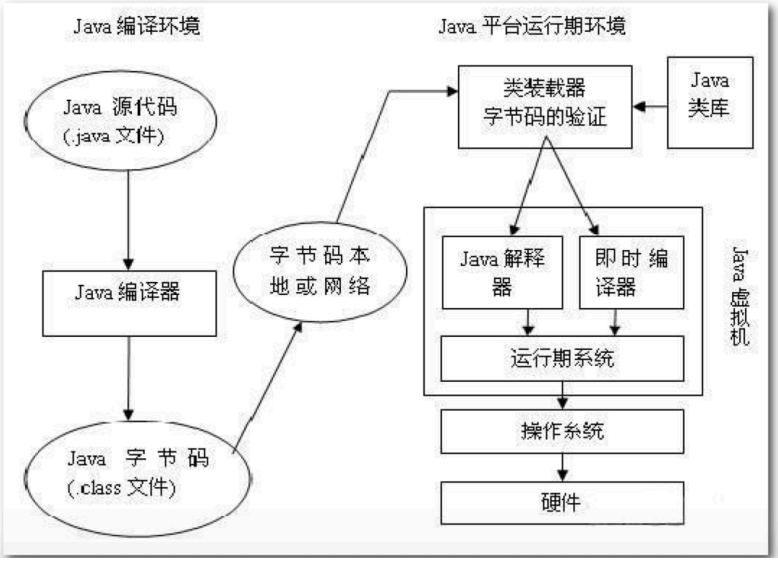
Java语言执行流程 :

-- **编译字节码** : Java编译器编译 .java源文件, 获得.class 字节码文件;

-- **装载类库** : 使用类装载器装载平台上的Java类库, 并进行字节码验证;

-- **Java虚拟机** : 将字节码加入到JVM中, Java解释器 和 即时编译器 同时处理字节码文件, 将处理后的结果放入运行时系统;

-- **调用JVM所在平台类库** : JVM处理字节码后, 转换成相应平台的操作, 调用本平台底层类库进行相关处理;



-- **签名规则** : (**参数1类型签名 参数2类型签名 参数3类型签名 参数N类型签名 ...) 返回值类型签名** , 注意参数列表中没有任何间隔;

Java类型 与 类型签名对照表 : 注意 boolean 与 long 不是大写首字母, 分别是 Z 与 J, 类是L全限定类名, 数组是[元素类型签名;

-- **类的签名规则** : **L + 全限定名 + ;** 三部分, 全限定类名以 / 分割;

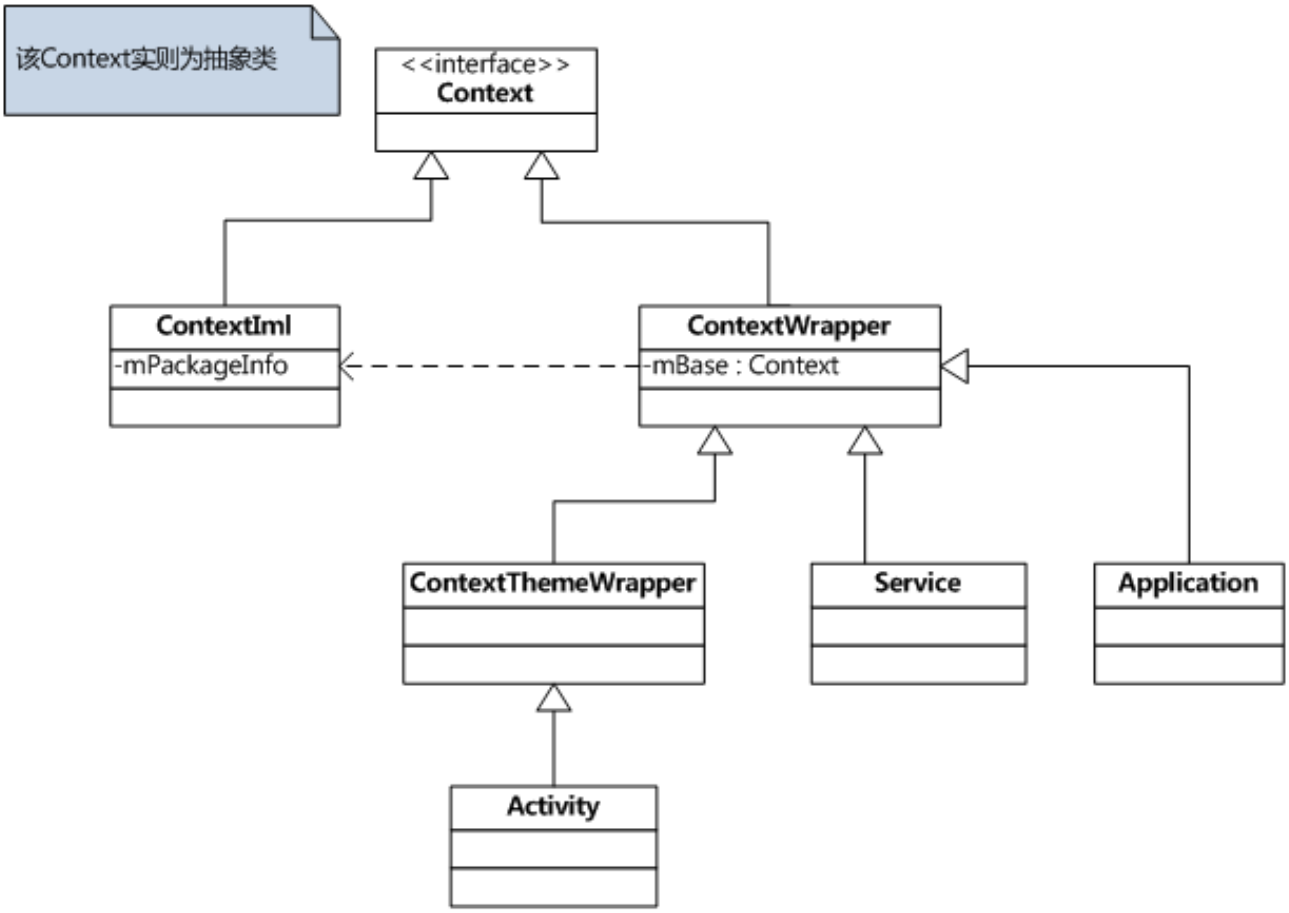
Java类型	类型签名
boolean	Z
byte	B
char	C
short	S
int	I
long	J
float	F
double	D
类	L全限定类名
数组	[元素类型签名

java 匿名类（转载） - henry_xu - 博客园
Android 命名规范 （提高代码可以读性） - 未来之路 的专栏 - 博客频道 - CSDN.NET
【Android测试工具】03. ApkTool在Mac上的安装和使用 - 码农老毕的学习笔记 - 博客频道 - CSDN.NET
Java中@Override的作用 - 海涛的CSDN博客 - 博客频道 - CSDN.NET
Eclipse快捷键 10个最有用的快捷键 - OPEN开源论坛
Android 中Odex文件生成与合并

####
android资源目录---assets与res/raw区别 - hshm20517的专栏 - 博客频道 - CSDN.NET
三个特殊的资源目录 /res/xml /res/raw 和 /assets
assets:用于存放需要打包到应用程序的静态文件，以便部署到设备中。与res/raw不同点在于，assets支持任意深度的子目录。这些文件不会生成任何资源ID，必须使用/assets开始（不包含它）的相对路径名。访问使用AssetManager类。
res/raw:存放通用的文件， 该文件夹内的文件将不会被编译成二进制文件，按原样复制到设备上。访问方式通过R类：((TextView)findViewById(R.id.txRaw)).setText(

readStream(getResources().openRawResource(R.raw.rawtext)));
/res/xml:这个目录中大家可能偶尔用到过，这里可以用来存储xml格式的文件，并且和其他资源文件一样，这里的资源是会被编译成二进制格式放到最终的安装包里的， 我们也可以通过R类来访问这里的文件，并且解析里面的内容。可以用XmlResourceParser xml = getResources().getXml(R.xml.data); 来访问数据。

####
Android中Context详解 ---- 你所不知道的Context - qinjuning、lets go - 博客频道 - CSDN.NET
1.Context概念可知:
1.1 它描述的是一个应用程序环境的信息，即上下文。
1.2 该类是一个抽象(abstract class)类，Android提供了该抽象类的具体实现类(后面我们会讲到是ContextIml类)。
1.3 通过它我们可以获取应用程序的资源和类，也包括一些应用级别操作，例如：启动一个Activity，发送广播，接受Intent信息 等。。
2.相关类继承关系：



3.应用程序在什么情况需要创建Context对象的？应用程序创建Context实例的情况有如下几种情况：

1、创建Application 对象时， 而且整个App共一个Application对象

2、创建Service对象时

3、创建Activity对象时

因此应用程序App共有的Context数目公式为：

总Context实例个数 = Service个数 + Activity个数 + 1（Application对应的Context实例）

####慕课网系列笔记

app-慕课网系列笔记

####
02.02
bindservice 服务不能unbind多次，否则会有异常。绑定源退出的时候需要unbind否则也会有异常
onServiceDisconnected
当启动源跟service的连接意外丢失的时候会掉用，比如service崩溃或被杀
但是当unbind时候是不会被调用到

####
11.21
所有的控件都有onclick事件，不只buton
text多个的时候实现跑马灯，参见安卓1 7-1
layout-weight 在各个控件是wrap-cpntent时候是正比于参数值，match-content时候是反比

####
11.21
assets 不会自动生成id，也不会自动占用空间

####
11.20
setjmp
sigsetjmp

####Camera的preview及takePicture大致过程

```java
public class MainActivity extends Activity {

    private SurfaceView cameraPreview;
    private Camera camera = null;
    private Callback cameraPreviewHolderCallback = new Callback() {

        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
            stopPreview();
        }

        @Override
        public void surfaceCreated(SurfaceHolder holder) {
            startPreview();
        }

        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int width,
                int height) {
            // TODO Auto-generated method stub

        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        cameraPreview = (SurfaceView) findViewById(R.id.cameraPreview);
        cameraPreview.getHolder().addCallback(cameraPreviewHolderCallback);

        findViewById(R.id.btnTakePic).setOnClickListener( new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                camera.takePicture(null, null,new Camera.PictureCallback() {

                    @Override
                    public void onPictureTaken(byte[] data, Camera camera) {

                        String path = null;
                        if ((path = saveTempFile(data)) != null) {

                            Intent i = new Intent(MainActivity.this,ImagePreviewAty.class);
                            i.putExtra("path", path);
                            startActivity(i);
                        } else {
                            Toast.makeText(MainActivity.this,"保存照片失败",Toast.LENGTH_SHORT).show();
```

```
                        }
                    }
                });
            }
        });
    }

    private String saveTempFile(byte[] bytes) {

        try {
            File f = File.createTempFile("img", "");
            FileOutputStream fos = new FileOutputStream(f);
            fos.write(bytes);
            fos.flush();
            fos.close();

            return f.getAbsolutePath();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return null;
    }

    private void startPreview() {
        camera = Camera.open();
        try {
            camera.setPreviewDisplay(cameraPreview.getHolder());
            camera.setDisplayOrientation(90);
            camera.startPreview();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void stopPreview() {
        camera.stopPreview();
        camera.release();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

####录音大致过程

```
    private MediaRecorder mr=null;

    private void startRecord(){
        if (mr==null) {

            File dir = new File(Environment.getExternalStorageDirectory(), "Sounds");
            if (!dir.exists()) {
                dir.mkdirs();
            }

            File soundFile = new File(dir, System.currentTimeMillis()+".amr");
            if (!soundFile.exists()) {
                try {
                    soundFile.createNewFile();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            mr = new MediaRecorder();
            mr.setAudioSource(MediaRecorder.AudioSource.MIC);
            mr.setOutputFormat(MediaRecorder.OutputFormat.AMR_WB);
            mr.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_WB);
            mr.setOutputFile(soundFile.getAbsolutePath());
            try {
                mr.prepare();
                mr.start();
            } catch (IllegalStateException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private void stopRecord(){
        if (mr!=null) {
            mr.stop();
            mr.release();
            mr = null;
        }
    }
```

####使用VideoView播放视频no2及no3

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_layout);

        videoview = (VideoView) findViewById(R.id.videoview);
        Uri uri = Uri.parse("android.resource://com.wilsonflying.playVideoByUri/"+R.raw.mingrisoft);

        videoview.setVideoURI(uri);
        videoview.requestFocus();
        videoview.start();

        videoview.setOnCompletionListener(new OnCompletionListener() {

            @Override
            public void onCompletion(MediaPlayer mp) {
                // TODO Auto-generated method stub

                Toast.makeText(getApplicationContext(), "播放完毕", Toast.LENGTH_SHORT).show();
                /****************重播***************/
//              videoview.start();
                /****************重播***************/

                /****************另外一种播放方式***************/
                Uri uri = Uri.parse(Environment.getExternalStorageDirectory().getPath()+"/Movies/mingrisoft.mp4");
                Intent  intent = new Intent();
                intent.setAction(Intent.ACTION_VIEW);
                intent.setDataAndType(uri, "video/mp4");

                try {
                    startActivity(intent);

                } catch (Exception e) {
                    System.out.println(e);
                    Toast.makeText(MainActivity.this, "ACTION_VIEW 重播失败", Toast.LENGTH_SHORT).show();
                    // TODO: handle exception
                }
                System.out.println("sdcard path:"+Environment.getExternalStorageDirectory().getPath());
                System.out.println("sdcard path:"+Environment.getExternalStorageDirectory());
                /****************另外一种播放方式***************/


            }
        });
    }
```

####使用VideoView播放视频no1

Accumulating_App

```
14  public class MainActivity extends Activity {
15
16      private VideoView videoView;
17      private File file;
18
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          // TODO Auto-generated method stub
22          super.onCreate(savedInstanceState);
23          setContentView(R.layout.layout_main);
24
25          videoView = (VideoView) findViewById(R.id.video);
26
27          String filePath = Environment.getExternalStorageDirectory()+"/Movies/mingrisoft.mp4";
28          file = new File(filePath);
29
30          MediaController mc = new MediaController(this);
31
32          if(file.exists()){
33              videoView.setVideoPath(file.getAbsolutePath());
34              videoView.setMediaController(mc);
35              videoView.requestFocus();
36              videoView.start();
37          }else{
38              Toast.makeText(this, "要播放的视频文件不存在", Toast.LENGTH_SHORT).show();
39          }
40
41          videoView.setOnCompletionListener(new OnCompletionListener() {
42
43              @Override
44              public void onCompletion(MediaPlayer mp) {
45                  // TODO Auto-generated method stub
46                  Toast.makeText(MainActivity.this, "视频播放完毕！", Toast.LENGTH_SHORT).show();
47              }
48          });
49      }
50  }
```

####MediaPlayer使用SurfaceView播放视频no2

Accumulating_App

```
public class MainActivity extends Activity {

    private SurfaceView surface;
    private MediaPlayer player;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_main);

        player = new MediaPlayer();
        surface =(SurfaceView) findViewById(R.id.surfaceView);
        final Button button_play = (Button) findViewById(R.id.button_play);
        final Button button_pause = (Button) findViewById(R.id.button_pause);
        Button button_stop = (Button) findViewById(R.id.button_stop);

        final String filePath = Environment.getExternalStorageDirectory()+"/Movies/mingrisoft.mp4";

        button_play.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                player.reset();
                try {
                    player.setDataSource(filePath);
                    player.setDisplay(surface.getHolder());
                    player.prepare();
                    player.start();
//                  surface.setBackgroundResource(R.drawable.bg_playing);//会遮挡视频画面

                    button_pause.setEnabled(true);
                    button_pause.setText("暂停");
                } catch (IllegalArgumentException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (SecurityException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IllegalStateException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
```

```
66
67          button_pause.setOnClickListener(new OnClickListener() {
68
69              @Override
70              public void onClick(View v) {
71                  // TODO Auto-generated method stub
72                  if(player.isPlaying()){
73                      player.pause();
74                      button_pause.setText("继续");
75                  }else{
76                      player.start();
77                      button_pause.setText("暂停");
78                  }
79              }
80          });
81
82          button_stop.setOnClickListener(new OnClickListener() {
83
84              @Override
85              public void onClick(View v) {
86                  // TODO Auto-generated method stub
87                  if(player.isPlaying()){
88                      player.stop();
89                      button_pause.setEnabled(false);
90                  }
91              }
92          });
93
94          player.setOnCompletionListener(new OnCompletionListener() {
95
96              @Override
97              public void onCompletion(MediaPlayer mp) {
98                  // TODO Auto-generated method stub
99                  Toast.makeText(MainActivity.this, "视频播放完毕！", Toast.LENGTH_SHORT).show();
100 //                 surface.setBackgroundResource(R.drawable.mpbackground1);//TODO:下次播放事后，该backgroud会遮挡画面，待查解决办法
101             }
102         });
103     }
104
105     @Override
106     protected void onDestroy() {
107         // TODO Auto-generated method stub
108         if(player.isPlaying()){
109             player.stop();
110         }
111         player.release();
112         super.onDestroy();
113     }
114 }
115
```

####MediaPlayer使用SurfaceView播放视频no1

```
15  public class MainActivity extends Activity {
16
17      private SurfaceView surfaceview;
18      private SurfaceHolder holder;
19      private Callback surfaceHolderCallback = new Callback() {
20          private MediaPlayer mp;
21
22          @Override
23          public void surfaceDestroyed(SurfaceHolder holder) {
24              // TODO Auto-generated method stub
25              mp.stop();
26              mp.release();
27          }
28
29          @Override
30          public void surfaceCreated(SurfaceHolder holder) {
31              // TODO Auto-generated method stub
32              mp = MediaPlayer.create(MainActivity.this, R.raw.mingrisoft);
33              try {
```

####MediaPlayer播放音乐大致过程

```
19  public class MainActivity extends Activity {
20
21      private MediaPlayer player;
22      private boolean isPause = false;
23      private File file;
24      private TextView textView;
25
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28          // TODO Auto-generated method stub
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.layout_main);
31
32          textView = (TextView) findViewById(R.id.text);
33          final Button button_play = (Button) findViewById(R.id.button_play);
34          final Button button_pause = (Button) findViewById(R.id.button_pause);
35          final Button button_stop = (Button) findViewById(R.id.button_stop);
36
37          File sdcard = Environment.getExternalStorageDirectory();
38          String filePath = sdcard.getAbsolutePath()+"/Music" +"/butterfly.mp3";
39          Log.d("shs", "path:"+filePath);
40
41  //      file = new File("/sdcard/tmp.mp3");
42          file = new File(filePath);
43          if(file.exists()){
44              player = MediaPlayer.create(getApplicationContext(), Uri.parse(file.getAbsolutePath()));
45          }else {
46              textView.setText("文件"+filePath+"不存在，无法播放");
47              button_play.setEnabled(false);
48              return;
49          }
50
51          player.setOnCompletionListener(new OnCompletionListener() {
52
53              @Override
54              public void onCompletion(MediaPlayer mp) {
55                  // TODO Auto-generated method stub
56                  play();
57                  Toast.makeText(getApplicationContext(), "播放完一次，重新开始播放", Toast.LENGTH_SHORT).show();
58              }
59          });
60
61          button_play.setOnClickListener(new OnClickListener() {
62
63              @Override
```

####SoundPool播放声音大致过程

####amlogic平台android7.1.2版本上编译apk直接install会失败提示签名不一致。关闭编译odex编译出来的版本就ok。

Failed to install out/target/product/U4C/system/app/GlobalSetting/GlobalSetting.apk: Failure [INSTALL_FAILED_UPDATE_INCOMPATIBLE: Package com.stv.globalsetting signatures do not match the previously installed version; ignoring!]

####有关属性动画
ObjectAnimator 控制动画的某个属性进行变化，与传统动画(帧动画、补间动画)不同的是即使位置改变后仍可以响应点击事件，而不是在原来位置上响应点击事件。
ValueAnimator 产生变化的数值。有其他逻辑控制其产生的数值再做展示或者别的应用。
有关帧动画、补间动画、属性动画更多详情见：Android 动画总结

####有关分辨率(名字与尺寸值对应关系)

####有关手势识别
1.使用GestureDetector、使用GestureOverlayView实现

####有关SharedPreference
一般只存基本数据，但也还是可以存放Object和图片
1.有关存放文件及目录，访问其他app数据九、使用SharedPreferences进行数据存储 - Ruthless - 博客园
2.有关获取SharedPreference的三种方式及默认文件名SharedPreferences的简单用法 - CSDN博客
3.有关存储Object和图片Android中最简单的数据存储方式：SharedPreferences
Activity.MODE_PRIVATE：表示该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容
Activity.MODE_APPEND：也是私有数据，新写入的内容会追加到原文件中

MODE_WORLD_READABLE(被其他应用读取)和MODE_WORLD_WRITEABLE(被其他应用写入)这两种模式已经在android 4.2版本以后废弃了
MODE_MULTI_PRIVATE：用于多个进程共同操作一个SharedPreferences文件

####activity和service互发消息
activity中通过绑定service的回调ServiceConnection中由binder创建Messenger
service中通过activity中发来的message中解析出activity的handler
详见：使用Handler实现Activity和Service之间的交互

####有关layout_weight
比如线性布局中垂直方向两个button，layout_height都是wrap_content，在layout_weight分别为2、1情况下则分别占屏幕的2/3 1/3
而如果layout_height是match_content，在layout_weight分别为2、1情况下则分别占屏幕的1/3 2/3
仅仅是button的layout_height不同，呈现效果两者相反

####模式鼠标及按键事件

- Instrumentation inst=new Instrumentation();
- inst.sendPointerSync(MotionEvent.obtain(SystemClock.uptimeMillis(),
- SystemClock.uptimeMillis(),
- MotionEvent.ACTION_DOWN, 240, 400, 0));
- inst.sendPointerSync(MotionEvent.obtain(SystemClock.uptimeMillis(),
- SystemClock.uptimeMillis(),
- MotionEvent.ACTION_UP, 240, 400, 0));

####重新学习内容汇总

1. 慕课网Java第一季：IMOOC--Java入门第一季笔记

####有关broadcast

Android总结篇系列：Android广播机制 - Windstep - 博客园

1.android:exported ——此broadcastReceiver能否接收其他App的发出的广播，这个属性默认值有点意思，其默认值是由receiver中有无intent-filter决定的，如果有intent-filter，默认值为true，否则为false。（同样的，activity/service中的此属性默认值一样遵循此规则）同时，需要注意的是，这个值的设定是以application或者application user id为界的，而非进程为界（一个应用中可能含有多个进程）

android:process ——broadcastReceiver运行所处的进程。默认为app的进程。可以指定独立的进程（Android四大基本组件都可以通过此属性指定自己的独立进程）

2.默认情况下，广播接收器也是运行在UI线程，因此onReceive方法中不能执行太耗时的操作。否则将因此ANR。一般情况下，根据实际业务需求，onReceive方法中都会涉及到与其他组件之间的交互，如发送Notification、启动service等。

全局广播存在安全性问题：

1)其他App可能会针对性的发出与当前App intent-filter相匹配的广播，由此导致当前App不断接收到广播并处理；

2)其他App可以注册与当前App一致的intent-filter用于接收广播，获取广播具体信息。

最常见的增加安全性的方案是：

1)对于同一App内部发送和接收广播，将exported属性人为设置成false，使得非本App内部发出的此广播不被接收；

2)在广播发送和接收时，都增加上相应的permission，用于权限验证；

3)发送广播时，指定特定广播接收器所在的包名，具体是通过intent.setPackage(packageName)指定在，这样此广播将只会发送到此包中的App内与之相匹配的有效广播接收器中。

4)使用本地广播

3.不同注册方式的广播接收器回调onReceive(context, intent)中的context具体类型

1).对于静态注册的ContextReceiver，回调onReceive(context, intent)中的context具体指的是ReceiverRestrictedContext；
2).对于全局广播的动态注册的ContextReceiver，回调onReceive(context, intent)中的context具体指的是Activity Context；
3).对于通过LocalBroadcastManager动态注册的ContextReceiver，回调onReceive(context, intent)中的context具体指的是Application Context。
注：对于LocalBroadcastManager方式发送的应用内广播，只能通过LocalBroadcastManager动态注册的ContextReceiver才有可能接收到（静态注册或其他方式动态注册的ContextReceiver是接收不到的）。
4.Android 3.1开始系统在Intent与广播相关的flag增加了参数，分别是FLAG_INCLUDE_STOPPED_PACKAGES和FLAG_EXCLUDE_STOPPED_PACKAGES。
FLAG_INCLUDE_STOPPED_PACKAGES：包含已经停止的包（停止：即包所在的进程已经退出）
FLAG_EXCLUDE_STOPPED_PACKAGES：不包含已经停止的包

####有关sharedUserId
进程间设置android:sharedUserId后可以共享资源
两个apk share相同的UserId必须是前面所用的private key相同

####2016.11.19手机截屏整理（时间逆序排列）

####

####有关AS报错Failed to resolve:xxxx
点击Install下载安装要求的东西即可。

####Ubuntu 安装AndroidStudio
sudo apt-add-repository ppa:paolorotolo/android-studio
sudo apt-get update
sudo apt-get install android-studio
另外可以从android studio上直接下载

####android自动化测试
Android自动化测试主要分为Monkeyrunner、Rubotium、UiAutomator、Monkey。
其中有关UiAutomator更多详情见：
[Android自动化测试（UiAutomator）简要介绍]
(http://blog.csdn.net/sasoritattoo/article/details/17579739)

####补充
command+shift+t 查找

####补充
屏幕适配：
1. android:anyDensity="true"应用程序安装在不同密度的终端上时，程序会分别加载xxhdpi、xhdpi、hdpi、mdpi、ldpi文件夹中的资源
2. 横屏/竖屏资源区分：drawable-hdpi、drawable-land-hdpi、drawable-port-hdpi；layout-port、layout-land；layout-640x360、layout-800x480
3. 自适应屏幕：
需要根据物理尺寸的大小准备5套布局，layout(放一些通用布局xml文件，比如界面中顶部和底部的布局，不会随着屏幕大小变化，类似windos窗口的title bar),layout-small(屏幕尺寸小于3英寸左右的布局)，layout-normal(屏幕尺寸小于4.5英寸左右)，layout-large(4英寸-7英寸之间)，layout-xlarge(7-10英寸之间)
需要根据dpi值准备5套图片资源，drawable，drawalbe-ldpi,drawable-mdpi,drawable-hdpi,drawable-xhdpi
4. 使用9-patch图片
[android中像素单位dp、px、pt、sp的比较]
(http://www.cnblogs.com/chiao/archive/2011/07/07/2100216.html)
[android 常见分辨率（mdpi、hdpi 、xhdpi、xxhdpi ）及屏幕适配注意事项]
(http://blog.csdn.net/sarsscofy/article/details/9249397)

####数据库专题

1. 命令行操作示例
insert into secure('name','value') values('location_providers_allowed','network');
update system set value='-2' where name='pointer_speed';
2. 数据类型：NULL、INTEGER、REAL、TEXT、BLOB。该数据类型是动态类型及弱引用，当某个值插入数据库中时，SQLite会检查它的类型，如果不匹配，SQLite会尝试将该值转换为该列的类型；如果不能转换，则该值将作为本身的类型存储；如果类型压根不支持则报错处理。
3. 如果Cursor的数据量特别大，特别是里边有blob信息时，应保证Cursor占用的内存计时释放掉，而不是等待gc来处理。如果等到gc来回收时，如果没有手动关闭掉系统会报错。

####Http专题
1. Get方式向服务器提交数据，需要对有中文的内容进行编码：url = url+"?name="+URLEncoder.encode(name, "utf-8") + "&age=" +age;

####Parcelable/Serializable
Serializable的作用是为了保存对象的属性到本地文件、数据库、网络流、rmi以方便数据传输，当然这种传输可以是程序内的也可以是两个程序间的
Parcelable的设计初衷是因为Serializable效率过慢，为了在程序内不同组件间以及不同Android程序间(AIDL)高效的传输数据而设计，这些数据仅在内存中存在，Parcelable是通过IBinder通信的消息的载体
代码实现方面：对于Serializable，类只需要实现Serializable接口，并提供一个序列化版本id(serialVersionUID)即可。而Parcelable则需要实现writeToParcel、describeContents函数以及静态的CREATOR变量，实际上就是将如何打包和解包的工作自己来定义，而序列化的这些操作完全由底层实现。

更多详情见：
[Android Parcelable和Serializable的区别]
(http://blog.csdn.net/djun100/article/details/9667283)
[Java 序列化的高级认识](http://www.ibm.com/developerworks/cn/java/j-lo-serial/index.html)

####Service
1. bindService方式，通过调用Context.bindService()启动，调用Context.unbindservice()结束，还可以通过ServiceConnection访问Service。在Service每一次的开启关闭过程中，只有onStart可被多次调用(通过多次startService调用)，其他onCreate，onBind，onUnbind，onDestory在一个生命周期中只能被调用一次。
2. Service.onBind如果返回null，则调用 bindService 会启动 Service，但不会连接上 Service，因此 ServiceConnection.onServiceConnected 不会被调用，但你任然需要使用 unbindService 函数断开它，这样 Service 才会停止。
3. 当客户端跟Service成功连接后会调用到onServiceConnected方法，但是当诸如Service崩溃或者被强行杀死之类的情况下跟Service的连接异常丢失的时候才调用到onServiceDisconnected方法。
4. Service也是运行在主线程中，所以service中做耗时操作还是需要创建线程的。
5. 为什么不能用Activity中创建的线程来替代service呢？这是因为Activity很难对Thread进行控制，当Activity被销毁之后，就没有任何其它的办法可以再重新获取到之前创建的子线程的实例。而且在一个Activity中创建的子线程，另一个Activity无法对其进行操作。但是Service就不同了，所有的Activity都可以与Service进行关联，然后可以很方便地操作其中的方法，即使Activity被销毁了，之后只要重新与Service建立关联，就又能够获取到原有的Service中Binder的实例。因此，使用Service来处理后台任务，Activity就可以放心地finish，完全不需要担心无法对后台任务进行控制的情况。
6. Service优先级高于Activity，但Service的系统优先级还是比较低的，当系统出现内存不足情况时，就有可能会回收掉正在后台运行的Service。如果你希望Service可以一直保持运行状态，而不会由于系统内存不足的原因导致被回收，就可以考虑使用前台Service。前台Service和普通Service最大的区别就在于，它会一直有一个正在运行的图标在系统的状态栏显示，下拉状态栏后可以看到更加详细的信息，非常类似于通知的效果。
[Android 中的 Service 全面总结]

(http://www.cnblogs.com/newcj/archive/2011/05/30/2061370.html)
[Service和Thread的关系](http://blog.csdn.net/jie1991liu/article/details/16105605)
Service和Thread的关系 - 期待 - 博客频道 - CSDN.NET

####HashMap、SparseArray、ArrayMap
HashMap：内部使用一个默认容量为16的数组来存储数据，数组中每一个元素存放一个链表的头结点，其实整个HashMap内部结构就是一个哈希表的拉链结构。HashMap默认实现的扩容是以2倍增加，且获取一个节点采用了遍历法，所以相对来说无论从内存消耗还是节点查找上都是十分昂贵的。
SparseArray：比HashMap省内存是因为它避免了对Key进行自动装箱（int转Integer），它内部是用两个数组来进行数据存储的（一个存Key，一个存Value），它内部对数据采用了压缩方式来表示稀疏数组数据，从而节约内存空间，而且其查找节点的实现采用了二分法，很明显可以看见性能的提升。
ArrayMap：内部使用两个数组进行数据存储，一个记录Key的Hash值，一个记录Value值，它和SparseArray类似，也会在查找时对Key采用二分法。
有了上面的基本了解我们可以得出结论供开发时参考，当数据量不大（千位级内）且Key为int类型时使用SparseArray替换HashMap效率高；当数据量不大（千位级内）且数据类型为Map类型时使用ArrayMap替换HashMap效率高；其他情况下HashMap效率相对高于二者。

####String/StringBuffer/StringBuilder
String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String。
StringBuffer 线程安全。更改内容的时候则是对对象本身进行操作，而不是生成新的对象，再改变对象引用。大部分情况下StringBuffer 优于String。
StringBuilder 非线程安全。StringBuffer的一个简易替换，用在字符串缓冲区被单个线程使用的时候。如果不考虑安全性，大部分情况下StringBuilder 优于 StringBuffer。

####BroadcastReceiver补充
1. 生命周期只有十秒左右，因此不能在BroadcastReceiver中做一些比较耗时的操作
2. 应该发送intent给Service，由Service来完成相应处理
3. 不能使用子线程，BroadcastReceiver生命周期结束后进程结束，其中的线程也就会被杀死
4. 动态注册广播优先级高于静态注册广播
5. 动态注册的广播需要在onDestroy中unregisterReceiver()掉
6. 普通广播：同级别接收先后是随机的、级别低的后收到广播、接收器不能截断广播也不能修改广播
7. 有序广播：同级别接收先后是随机的、能截断广播(abortBroadcast)、也能修改广播(setResultExtras(Bundle bundle)后，后收到广播的接收器可以getResultExtra获取数据)
8. 粘滞广播：可以先发送，再注册

####ContextMenu/OptionMenu
OptionMenu对应的是activity，一个activity只能拥有一个选项菜单
ContextMenu对应的是View，每个View都可以设置上下文菜单

reviewwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww

####OOM
1. 即out of memory，使用的内存量超过了vm分配给app的最大值
2. 多数原因是因为使用了太多bitmap或者分配大数组
3. 有挂bitmap使用上的优化：
    * 只加载可见区域的bitmap
    * 在滑动过程中不加载，滑动停止后再加载
    * 使用软引用机制，在内存不足的时候允许gc回收bitmap占用的内存
    * 调用recycle()通知gc尽快回收不再使用的内存
    * 调整图像大小，手机屏幕尺寸有限，有时图像大小可以做适当调整、对图片进行压缩
    * 采用低内存占用量的编码方式，比如Bitmap.Config.ARGB_4444

* 使用LruCache、DiskLruCache进行缓存

更多详情见：
[Android Out Of Memory(OOM) 的详细研究]
(http://www.cnblogs.com/wanqieddy/archive/2012/07/18/2597471.html)
[Android处理图片OOM的若干方法小结](http://www.2cto.com/kf/201208/148379.html)
[Android OOM 问题的总结](http://my.oschina.net/line926/blog/271175)

####内存泄露
1. 判断一个内存空间是否符合垃圾收集标准有两个：一个是给对象赋予了空值null，以下再没有调用过，另一个是给对象赋予了新值，这样重新分配了内存空间。
2. 导致内存泄露的常见几个场景：
* 静态集合类引起的内存泄露：静态变量的生命周期跟应用程序一致，只将加载到集合中的Object赋值为null还不够，需要先从集合中删除；此外也可以直接将集合类赋值为null
* 当集合里面的对象属性被修改后，再调用remove方法时不起作用
* 监听器的使用：带有监听器的对象释放时候需要删除掉相应的监听器
* 各种连接，诸如数据库连接dataSourse.getConnection()、socket连接、io等，需要即时关掉。有关Resultset 和Statement 的更多详情，见下面连接。
* 大量临时变量的使用
* 内部类的使用容易被忽略掉释放
* 外部模块的引用，A类中调用了B类的一个方法，参数中带了B的一个对象，这时B中就保持了对该对象的引用，需要留意B是否提供了相应的方法去除该引用。
* 单例模式：单例对象在初始化后将在应用程序的整个生命周期中存在，如果单例对象中持有外部对象的引用，这个对象将一直得不到释放。
更多详情见：
[Java内存泄露原因详解](http://blog.csdn.net/seelye/article/details/8269705)
[Java的内存泄漏](http://www.ibm.com/developerworks/cn/java/l-JavaMemoryLeak/)

####多继承
1. Java中是否支持多继承是个语言描述问题。Java中只支持类的单继承，接口之间的继承

同样也是使用extends关键字，但是接口之间是支持多继承的
2. 如果在两个父接口中分别定义了名称和参数都相同，而返回结果却不同的方法，这时会出现编译问题。方法的重载只能是相同的方法名，不同的输入参数；如果两个方法具有相同的方法名，相同的输入参数，只是不同的返回参数，是不能作为重载方法的，所以对于编译器来说，这里是一个方法的重复定义，明显是不能通过编译的。这样的问题也存在于一个类同时实现多个接口的情况，所以，在这些情况下，我们必须注意一点，就是具有相同方法名，相同输入参数的方法，是不能出现在同一个类或接口中的。

####多线程专题
1. Runnable方式相较于继承Thread方式创建线程有两个优点：1) Runnable方式可以避免Thread方式由于Java单继承特性带来的缺陷 2) Runnable的代码可以被多个线程共享，适合于多个线程处理同一资源的情况。
2. 有关线程同步：
    * synchronized 修饰代码块
    * synchronized 修饰(普通/静态)方法
    * volatile 修饰变量，volatile修饰域相当于告诉虚拟机该域可能会被其他线程更新，因此每次使用该域就要重新计算，而不是使用寄存器中的值，volatile不会提供任何原子操作，它也不能用来修饰final类型的变量
    * 使用ReenreantLock/ReentrantReadWriteLock，new出对象后即可使用lock及unlock方法来获取/释放锁，需要留意及时释放锁，避免出现死锁情况，通常在finally代码释放锁。此外还有trylock/lockInterruptiblynewCondition等细分用法。更多详情见[Lock和Condition](http://luan.iteye.com/blog/1849712)
    * wait/notify 机制
    * ThreadLocal 每一个使用该变量的线程都获得该变量的副本， 副本之间相互独立，这样每一个线程都可以随意修改自己的变量副本，而不会对其他线程产生影响。其存放的值是线程内共享的，线程间互斥的，主要用于线程内共享一些数据，避免通过参数来传递。更多有关get/set/remove/initialValue的使用说明详情参加[Java线程(篇外篇)：线程本地变量ThreadLocal](http://blog.csdn.net/ghsau/article/details/15732053)

####多线程之wait/notify
1. 在任一时刻，对象的控制权智能被一个线程拥有
2. wait(),notify(),notifyAll()不属于Thread类,而是属于Object基础类,也就是说每个对像都有wait(),notify(),notifyAll()的功能
3. 要执行wait/notify/nofiyAll这三个方法，须当前线程取得了该对象的控制权，否则会报java.lang.IllegalMonitorStateException
4. 线程要取得对象的控制权方法有三：1) 执行对象的某个同步实例方法 2) 执行对象的对应类的同步静态方法 3) 执行对该对象加同步锁的代码块。
5. 在调用wait的时候，线程自动释放其占有的对象锁，同时不会去申请对象锁。当线程被唤醒的时候，它才再次获得了去获得对象锁的权利。调用完wait，该线程就已经不是currentthread了。
6. 当B调用notify/notifyAll的时候，B正持有obj锁，因此，A1,A2,A3虽被唤醒，但是仍无法获得obj锁。直到B退出synchronized块，释放obj锁后，A1,A2,A3中的一个才有机会获得锁继续执行。
7. notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。notifyAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。
8. 同步分为类级别和对象级别，分别对应着类锁和对象锁。类锁是每个类只有一个，如果static的方法被synchronized关键字修饰，则在这个方法被执行前必须获得类锁；对象锁类同。

####"xxxx" is not translated in "en" (English)及"xxx" is translated here but not found in default locale问题
确实是在res/values/strings.xml中定义的一些值在values-en中没有定义，不过如果想忽略的话，可以在windows-preferencs--Android--Lint Error Checking--missing translation 有fetal改成warning即可；后一个问题相应的修改Extra Translation由fetal到warning即可。

####fragment篇
1. 静态加载：在布局文件中定义一个fragement，指定id/tag以及fragment实现类
2. 动态添加：需要用到Fragment事务(对Fragment进行添加、移除、替换以及执行其他的

一些动作，提交给activity的每一套变化成为一个事务)，在Fragment事务中add一个fragment到一个layout中。
3. 如果需要在按下back按键时返回到前一个fragment状态，调用commit之前需要addToBackStack。
4. 静态加载一个fragment后，fragment中的控件对activity是可见的，可以通过findViewById的方式找到并操作fragment中控件。
5. 声明周期：见下图
6. Fragment与Activity通信
fragment中调用getActivity获取它所在的activity，activity中通过FragmentManager的findFragmentbyId或者findFragmentByTag获取fragment。
Acitivity—>Fragment：在Activity中创建Bundle，作为Fragment的setArguments方法的参数。
Fragment—>Activity：在Fragment中定义接口，包含该Fragment的Activity中实现该回调接口，这样在Fragment中可以调用该接口传递数据到Activity。

####Handler篇
1. 设计Handler机制根本目的是解决更新UI时多线程并发的问题。
2. Handler：1.发送Message或者Runnable对象 2. 处理接收的消息
3. Looper：内部包含一个MessageQueue，Handler发送的消息会被加入的MessageQueue中；Looper.loop()方法则是一个死循环，负责从MessageQueue中读取消息进行处理，在处理过程中会调用到Handler的dispatchMessage方法，该方法中没有被Callback截获的话就能调用到Handler的handleMessage方法了。
4. MessageQueue：一个存储消息的队列，在Looper对象中进行创建
5. 主线程中创建Handler，该Handler的handleMessage是运行在主线程的，也就不能做一些耗时操作。
6. 在新建的线程中new Handler的话需要首先调用Looper.prepare()方法来创建Looper及MessageQueue，并在创建Handler之后调用Looper.loop()方法启动循环检测。此Handler的handleMessage将是在所属线程中运行。
7. Handler的post系列接口，Runnable对象的run方法是在UI线程中运行的；通过removeCallbacks(Runnable r)的方法移除一个消息；在new一个Handler对象时候创建

Callback参数，可在该Callback对象中的handleMessage(带boolean返回值)中return true达到截获消息的效果。
8. 可以在主线程创建Handler的时候传入一个线程的Looper对象作为参数，从而实现子线程中处理消息的目的。在获取一个线程的Looper对象传入Handler时候存在并发问题，因此最好使用HandlerThread来创建这个消息吹线程。
9. HandlerThread是一个封装了Looper的线程，在getLooper中做了同步处理，可以确保返回的不为null。

####更新UI的几种方式
1. handler post出来的Runnable对象中更新
2. handler sendMessage后在handler中的handleMessage方法中更新
3. runOnUiThread(Runnable action)方法中Runnable对象中更新
4. view的post(Runnable action)方法中Runnable对象中更新

####AndroidAnnotation篇
1. eclipse安装插件'Eclipse java development tools'(太特么的慢了，翻墙环境下三个小时以上)，才会在右键--properties—Java Compiler中出现'Annotation Processing'，详情见[android eclipse 没有Annotation Processin选项](http://blog.csdn.net/caiwenfeng_for_23/article/details/38959685)
2. 由于会继承当前SomeActivity类生成SomeActivity_子类，然后生成本该我们自己写的代码，所以成员变量和成员方法不能定义成private。
3. 一个app最多支持65536个方法，在大型项目中使用注解框架有可能出出现方法数超限。

####调用一些sdk中类而找不到的问题
创建工程默认提供的android.jar可能会有些类没有包含，需要将sdk中的一些jar引入到工程中才能编译过。比如修改静默安装apk问题时候测试代码中 IPackageInstallObserver, IPackageDeleteObserver 解析不到，将sdk中的 out/target/common/obj/JAVA_LIBRARIES/framework_intermediates/classes.jar 添加到工程中，而且需要调整优先级比工程中默认提供的android.jar包优先级高，方法Build Path--Configure Build Path--Order and Export中调整。

####静默安装apk篇
1. 达到静默安装的目的
普通安装方式：Intent intent = new Intent(Intent.ACTION_VIEW);intent.setDataAndType(Uri.fromFile(new File(strApk)), "application/vnd.android.package-archive");startActivity(intent);
静默安装：调用PackageManger的接口。pm.installPackage(mPackageURI, observer, PackageManager.INSTALL_REPLACE_EXISTING, Intent.EXTRA_INSTALLER_PACKAGE_NAME); 需要实现一个PackageInstallObserver()实例observer。
静默卸载：getPackageManager().deletePackage("com.wilsonflying.testframelayout", observer_delete, 0);
特殊点：需要系统签名
2. 安装apk的调用者一般是应用商城的角色，应用商城需要自身的频繁升级的，如果是第三方发布的话又不能持有系统签名的key。需要支持非系统签名的应用商城的调用installPackage接口。
安装流程会走到PackageManagerService的installPackageWithVerificationAndEncryption()方法，其中首先会有个mContext.enforceCallingOrSelfPermission(android.Manifest.permission.INSTALL_PACKAGES, null);这里会根据调用这的pid、uid进行权限的鉴别，如果没有权限抛异常出来，一路传上去到调用安装的apk。
　目前在installPackageWithVerificationAndEncryption()中获取调用者的包名，如果在允许范围内则不作上述权限鉴别动作，流程继续走下去能够安装成功。

####使用Genymotion调试出现错误INSTALL_FAILED_CPU_ABI_INCOMPATI
调试极光推送的demo时候，无法启动，报如上错误，下载Genymotion-ARM-Translation.zip拖到Genymotion中即可。

####内存泄露查找工具LeakCanary

####AS快捷键之Windows篇
F2 定位出错处
alt+enter 错误说明
ctrl+d 复制行
ctrl+y 删除行，可以用ctrl+x (在不选中的情况下为剪切行，可以一举两得)
ctrl+alt+L 格式化代码(会提示格式化了多少行，及ctrl+alt+shift+L 更多格式化场景)

####
(1)Android-SDK 命令行创建AVD模拟器_小海豚_新浪博客

####有关Android studio
1. 安装篇
Windows下需要配置支持HAXM，具体参照 [安装Intel HAXM为Android 模拟器加速](http://www.tuicool.com/articles/meQbmmb)，还算比较顺利，只不过在配置BIOS中开启虚拟加速机制的时候找了半天。我这戴尔台式机，位置是Overclocking--CPU Features--Intel Virtualization Tech,选中为Enabled，F10保存退出即可。
号称30秒启动的虚拟机，基本没有改善，而且还真真的占用了2G内存。还是上Genymotion，业界良心。
2. AndroidStudio+Genymotion
装下插件即ok，Settings--Plugins--搜索Genymotion--下方的按钮Browse repositories在线安装--弹出界面选择Install。
然后工具栏新出现一个Genymotion Device Manager中配置Genymotion安装路径：D:\Program Files\Genymobile\Genymotion。
3. 导入之前的项目有可能会报'找不到API 19 sdk platform'的错
AS默认只带了API22的platform文件，所有到sdk manager中将常见的api 21/19/17/15安装一下，有关SDK Manager设置代理见之前笔记'Eclipse之sdk manager篇’
Mac上安装完后查看sdk Manager居然自动找到了Eclipse下载过的sdk路径，不用重新下

载，爽歪歪。

4. 导入之前的项目时候，出现如下报错

将MyApplication中build.gradle中的android字段删掉即可，记得点击下'Sync Project with Gradle Files'即可生效：

5. Mac上安装AS，下载sdk阶段一直报错'The following SDK components were not installed: sys-img-x86-addon-google_apis-google-22 and addon-google_apis-google-22’

有全局翻墙环境，但是点击retry仍然一直失败，点击cancel然后就退出了，重新打开AS一路next下来又重复上述问题。

直到重新打开AS，在Welcome界面直接点击cancel而不是点击next，则出现了创建Android App工程的界面，终于能用了。

6.ubuntu上遇到奇怪的打印

首先查看compile sdk version 和build tools version 是否匹配，再看跟build.gradle中的dependencies中compile 'com.android.support:appcompat-v7:23.4.0'不匹配。如上图中改为21.+即可。

7.启动as，一直提示不能启动adb(如下图)，运行的时候表现为不能找到模拟器。可能是eclipse占用了adb，确认eclipse没开后，重新配置genymotion 的ADB选项从默认改为使用Android sdk解决问题。

8.从2.1升级到3.0后，as中之前能编译通过的工程也报错：

Error:java.lang.NullPointerException (no error message)，在module settings中提示更换java 8.

更换jdk为java8后原来的工程正常了，升级后新建的工程还有报错。

####反编译

反编译：java -jar ~/tools/apktool_2.0.0.jar d myhw/ apks/myhw.apk

重新打包：java -jar ~/tools/apktool_2.0.0.jar b myhw/ myhw_new.apk

反编译：java -jar tools/apktool1.5.2.jar d apks/myhw.apk apks/myhw

出现报错：Could not decode arsc file

原因：apktool工具版本太老，更新新版本的apktool

sublime工具查看反编译出来的smali
重新打包：java -jar tools/apktool1.5.2.jar b apks/myhw new_myhw.apk
出现报错：Could not run progress “aapt"
原因：需要将adt—sdk—build-tools—21.1.2(获取其他某个版本)—aapt的路径添加到环境变量PATH中
对apk签名：java -jar tools/apk-signer-1.8.5.jar
选择了待签名apk、key文件进行签名时出现如下乱码，待分析。还是用Linux中签名方式：java -jar $build_path/linux-x86/framework/signapk.jar $build_path/security/platform.x509.pem $build_path/security/platform.pk8 $input_file $output_file

####不能打开包含在apk里边的数据库，如需使用得先写到data相应目录中或者sdcard中

####Eclipse之sdk manager篇
如果没有全局的翻墙环境，可以配置sdk manager中代理。Android SDK Manager--tools--options，填写HTTP Proxy Server及Port信息。
mirrors.neusoft.edu.cn:80 或者 mirrors.opencas.cn:80。
更多镜像地址及Android开发工具下载可见：[AndroidDevTools简介](http://www.androiddevtools.cn/)

####有关AsyncTask
1. 需要在UI线程中创建AsyncTask的实例以及调用AsyncTask的execute方法
2. 重写的四个方法是系统自动调用的，不能手动去调用
3. 每个AsyncTask只能被执行一次，多次调用会引发异常
4. 只有doInBackgroud方法是运行在其他线程做异步操作，其他三个方法都是运行在UI线程，也就都可以操作UI
5. AsyncTask的cancel只是将对应的AyncTask标记为cancel状态，并不是真正的取消线程的执行。需要在doInBackgroud和onProgressUpdate里边检测线程是否isCancelled状态，然后做相应操作。

6. publishProgress是一个final类型的，无法被Override
7. excute的实现代码中有个THREAD_POOL_EXECUTOR，就是一个活生生的线程池，创建了一个大小为CORE_POOL_SIZE的线程池，这个的大小是多大？默认它是通过获取当前CPU的核数，根据CPU的核数来计算的，也就是说，当CPU核数小于4的时候，线程池为2，当CPU核数大于4个的时候，线程池的默认大小为4。
8. AsyncTask的实例必须在UI thread中创建，并且execute方法必须在UI thread中调用，但其实这也是一个缺点，如果不在UI线程中创建，那onPreExecute()方法执行就是在子线程中了，但是往往onPreExecute方法很多业务逻辑都在UI线程。
9. AsyncTask不仅可以有效管理线程数量，任务排序调度，还能实现线程之间的通信，还有有效解决android中的ANR，以及UI界面的更新问题，AsyncTask做了这一些列的封装。
10. 设计初衷/优点：AsyncTask不仅可以有效管理线程数量，任务排序调度，还能实现线程之间的通信，还有有效解决android中的ANR，以及UI界面的更新问题。

####ndk
1. 环境搭建
    * 下载ndk，地址见[AndroidDevTools简介](http://www.androiddevtools.cn/)
    * chmod a+x android-ndk-r10d-darwin-x86_64.bin; ./android-ndk-r10d-darwin-x86_64.bin
    * 添加ndk-build所在目录到环境变量PATH

2. 生成头文件
终端命令行方式，在工程根目录下：
javah -classpath bin/classes -d jni -jni com.wilsonflying.testjni.AccessNative
javah -classpath bin/classes -d jni com.wilsonflying.testjni.AccessNative
配置eclipse自动生成头文件：
[在eclipse中快速开发JNI，一键生成C头文件.h，以及一键使用NDK交叉编译]
(http://www.oschina.net/question/1402563_133543)
3. ndk-build
终端命名行方式，在工程根目录下直接执行：
ndk-build

配置eclipse自动执行：
[在eclipse中快速开发JNI，一键生成C头文件.h，以及一键使用NDK交叉编译]
(http://www.oschina.net/question/1402563_133543)
4. 在preference中没有NDK的配置项的问题
目前adt bundle版本是adt-bundle-mac-x86_64-20140702，在preference中没有NDK配置项。打开前不久的旧版本adt-bundle-mac-x86_64-20131030中的eclipse是有NDK配置项的。将~/Applications/adt-bundle-mac-x86_64-20131030/eclipse/plugins/com.android.ide.eclipse.ndk_22.3.0.v201310242005-887826.jar 拷贝到目前版本对应plugins目录，重启eclipse就ok了。
5. 配置NDK后在.C .CPP文件中仍然不能自动补齐的问题
需要在项目开始写jni代码前通过Add Native Support的方式自动生成jni目录及默认的cpp文件和Android.mk。方法是右键工程名称—Android tools—Add Native Support。这个时候在去编辑代码就可以自动补齐，并且能够跟编辑java代码时一样实时检查代码语法错误。

####有关JNI找不到android.app.Activity的类文件
javah -classpath bin/classes:/Users/cindy/Applications/adt-bundle-mac-x86_64-20140702/sdk/platforms/android-21/android.jar -
d jni com.wilsonflying.testjni2.MainActivity
通过在Activity中声明native的方法，也是可以通过javah生成头文件的。只不过需要在制定classpath路径中添加android.jar所在目录，否则会报错提示找不到android.app.Activity的类文件。这样还是太繁琐，而且生成的头文件里边会有非常多没有用的东西，所以还是单独创建一个类，在其中声明native的方法，然后生成头文件。

####android:protectionLevel
normal 表示权限是低风险的，只要申请了就可以使用，安装时不需要用户确认
dangerous 表示权限是高风险的，系统将可能要求用户确认，才会授予此权限
signature 只有当应用程序所用数字签名与声明此权限的应用程序所有数字签名相同时，才能将权限授给它
signatureOrSystem 签名相同，或者申请权限的应用为系统应用

另外一个是android:permissionGroup属性，表示一个权限组。可以将权限放在一个组中，但对于自定义权限，应该避免设置此属性。如果确实希望设置此属性，可以使用以下属性代替：android.permission-group.SYSTEM_TOOLS。

####有关启动模式launchMode
1.standard 不管有没有已存在的实例，都生成新的实例
2.singleTop 如果发现有对应的Activity实例正位于栈顶，则重复利用，不再生成新的实例
3.singleTask 如果发现有对应的Activity实例，则使此Activity实例之上的其他Activity实例统统出栈，使此Activity实例成为栈顶对象，显示到幕前
4.singleInstance 这种启动模式比较特殊，因为它会启用一个新的栈结构，将Acitvity放置于这个新的栈结构中，并保证不再有其他Activity实例进入
还有相对应的FLAG详情见：
[Activity Intent相关FLAG介绍](http://blog.sina.com.cn/s/blog_6f3ff2c90101j50x.html)
5.taskAffinity属性不对standard和singleTop模式有任何影响，standard和singleTop启动模式都是在原任务栈中新建Activity实例，不会启动新的Task，即使你指定了taskAffinity属性。
那么什么是taskAffinity属性呢，可以简单的理解为任务相关性。
* 这个参数标识了一个Activity所需任务栈的名字，默认情况下，所有Activity所需的任务栈的名字为应用的包名
* 我们可以单独指定每一个Activity的taskAffinity属性覆盖默认值
* 一个任务的affinity决定于这个任务的根activity（root activity）的taskAffinity
* 在概念上，具有相同的affinity的activity（即设置了相同taskAffinity属性的activity）属于同一个任务
* 为一个activity的taskAffinity设置一个空字符串，表明这个activity不属于任何task
指定方式如下：<activity android:name=".ActivitySingleTop"
android:launchMode="singleTop" android:taskAffinity="com.castiel.demo.singletop"/>

####关启动模式launchMode
基础总结篇之二：Activity的四种launchMode - scott's blog - 博客频道 - CSDN.NET

Activity Intent相关FLAG介绍_一切依旧_新浪博客

1.Intent.FLAG_ACTIVITY_NEW_TASK

默认的跳转类型,它会重新创建一个新的Activity，不过与这种情况，比如说Task1中有A,B,C三个Activity,此时在C中启动D的话，如果在AndroidManifest.xml文件中给D添加了Affinity的值和Task中的不一样的话，则会在新标记的Affinity所存在的Task中压入这个Activity。如果是默认的或者指定的Affinity和Task一样的话，就和标准模式一样了启动一个新的Activity.

如果试图从非activity的非正常途径启动一个activity，比如从一个service中启动一个activity，则intent比如要添加FLAG_ACTIVITY_NEW_TASK 标记。

2.FLAG_ACTIVITY_NO_USER_ACTION

onUserLeaveHint()作为activity周期的一部分，它在activity因为用户要跳转到别的activity而要退到background时使用。比如,在用户按下Home键，它将被调用。比如有电话进来（不属于用户的选择），它就不会被调用。

那么系统如何区分让当前activity退到background时使用是用户的选择?

它是根据促使当前activity退到background的那个新启动的Activity的Intent里是否有FLAG_ACTIVITY_NO_USER_ACTION来确定的。

注意：调用finish()使该activity销毁时不会调用该函数

####setImageDrawable与setImageResource

ImageView 使用：

```
imageview1.setImageDrawable(getResources().getDrawable(R.drawable.shoe_default));
```

ImageSwitcher使用：

```
private int[] imageId = new int[] { R.drawable.img01, R.drawable.img02};
imageSwitcher.setImageResource(imageId[index]);
```

####scrollview与listview

ScrollView继承自FrameLayout，是一个Layout！可以滚动显示一个占据的空间大于物理显示的视图列表。scrollview只能包含一个子视图或试图组，在实际使用中通常包含的是一个垂直的LinearLayout。在页面展示后，scrollview中的所有内容都加载到了内存中，当数据量大时会比较耗资源。与之对应的还有个HorizontalScrollView，可以在水平方向滚动视

图。
ListView 继承自AbsListView，是一个可以在垂直方式滚动显示view视图的列表。只有在滑动到显示界面才会加载之前没有显示的内容，数据量大的时候性能方面优势比较明显。可以精确到其中每一个item的点击事件。使用ListView有两种方式，一种是通过一个继承了ListActivity的Activity，在其中设定ListAdapter，对于这种方式，比较适用于整个页面就是一个ListView；第二种方式就是直接使用ListView控件。

####空进程
运行这些进程的唯一原因是作为一个缓存，缩短下次程序需要重新使用的启动时间。系统经常中止这些进程，以调节程序缓存和系统缓存的平衡
更多有关进程分类、进程生命周期、Android内存管理信息见：
[Android 进程生命周期 Process Lifecycle]
(http://www.cnblogs.com/mengdd/p/3139934.html)
[Android内存管理机制详解](http://blog.csdn.net/chaihuasong/article/details/8289367)

####软引用
如果一个对象只具有软引用，则内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。
其特点是它的一个实例保存对一个对象的软引用，该软引用的存在不妨碍垃圾收集线程对该Java对象的回收。也就是说，一旦SoftReference保存了对一个Java对象的软引用后，在垃圾线程对这个Java对象回收前，SoftReference类所提供的get()方法返回Java对象的强引用。另外，一旦垃圾线程回收该Java对象之后，get()方法将返回null。
更多有关软引用及强引用、弱引用、虚引用见：
[android使用软引用构建缓存](http://blog.csdn.net/hbzh2008/article/details/9038029)
[android 解决图片大量上载：软引用必须懂4点]
(http://www.educity.cn/wenda/176212.html)

####有关listActivity的id
如果只是普通的文本内容的listview，完全可以不定义layout。如果需要定制布局，在布局

文件中添加listview的时候需要将其id设置为android:id="@android:id/list"，或者android:id="@id/android:list”，否则会有这种报错：Your content mush have a listView whose id attribute is 'android.R.id.list'

####raw中命名文件不能有大写，asset中文件名可以有大写，什么破规矩！

####有关Java内部类引用外部变量需要定义为final
根据创建位置，内部类可以分为成员的、方法的、匿名的、接口的。
对于成员内部类来说，会持有一份外部类当前对象的引用：Out.this。这样就可以调用外部类可见的方法及成员变量，调用方法是通过持有的外部类当前对象的引用。
对于方法内部类来说，通过外部类当前对象的引用访问不到所在方法中的局部变量。Java设计的解决办法是：将局部变量复制一份给内部类使用。方法则是在内部类初始化的时候通过构造方法传值的方式。这样引出一个问题就是，**方法中变量与内部类中变量副本的一致性问题**。解决此问题的方法就是需要将方法中的局部变量定义为final！
更多详情见：[Java内部类与final关键字详解]
(http://blog.csdn.net/ygj281583295/article/details/9059737)

####安全篇
1.
2. 对称加密
置换加密、转置加密、乘积加密
DES/AES
高效、密钥交换的问题、不如RSA加密的安全程度高，但256bit的AES足够绝大多数安全领域
3. 非对称加密
RSA
安全性足够高、没有密钥交换的问题、效率低，对大数据加密很慢
4. 密钥交换
非对称加密方式交换密钥
Diffi e-Hellman密钥交换协议

5. HASH(哈希、散列、数字摘要、数字指纹、MD),变长输入变换成定长输出
常见hash算法：MD5(128bit)、SHA1(160bit)
易变性：即使原数据发生1bit的变化，hash的输出将发现不可预知的巨大变化
不可逆：通过hash结果构造出输入信息是不可能的或者说极其困难
应用场景：防篡改、防损坏、认证
HMAC(Hash-based Message Authentication Code)，使用key对原始信息进行变换后再进行HASH。解决：1.尾部直接附带消息摘要的问题(篡改内容的同时篡改消息摘要)2.直接对密码做HASH传输的认证问题(重放攻击，不用破解，截获或直接使用。(预防重放攻击，还可以从server端先发送随机码，client段将密码和随机码拼接然后做HASH))。
6. 数字签名

f

t

7. 证书
可完成公钥的存储和交换；利用签名保护数字证书本身；

####Effective UID/ Real UID

Linux中，父子进程间UID的世袭原则：身份世袭而权利不世袭，及子进程可以继承Real UID,不能继承Effective UID。表现为子进程的Real UID=子进程的Effective UID=父进程的Real UID。
典型的实例:passwd, 运行时候Effective UID被提升为root，但RUID不会变。su，EUID和RUID均会改变。

有了setuid 还有setgid及对应的Effective GID、Real GID。
android_filesystem_capability.h中定义了capability的描述。

t

t

t

P’:子进程，P:父进程，F:可执行文件的Capability

t

t

####FLAG_ACTIVITY_CLEAR_TOP
A,B,C,D依次启动了四个Activity，由D到B而且不再需要回到C，则可以在D中启动B的时候加入intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
引申：由D退出程序，则可以用如上方法启动A，同时配置A为android:launchMode="singleTask"，这时启动A就不会调用oncreate(),而是响应onNewIntent()，在A中重写onNewIntent()添加finish动作，这时将会退出整个程序。
更多详情见：[ActivityGroup相关--getLocalActivityManager() 以及intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)用法](http://blog.csdn.net/getchance/article/details/8444589)

####定制控件几种方式
1. 继承其他空间类(EidtText、Button)
2. 组合方式。当前空间类从容器里继承，并将若干控件添加到当前的容器中
3. 绘制控件，也就是空间类从View继承，并从onDraw方法中从零绘制控件。
编写控件类的时应将可能变化的值通过属性设置，而不是直接固化在控件类中。

####getApplicationContext与this
对于一般的Android应用，通常有两种Context，即Activity和Application。
getApplicationContext() 返回应用的上下文，生命周期是整个应用，应用摧毁它才摧毁。
Activity.this 返回当前activity的上下文，生命周期只是它所在的Activity，activity 摧毁它就摧毁。
如果是在匿名内部类中，this通常就是不能用的，因为内部类所new出来的对象的生命周期比较短。

####layout_gravity/gravity/padding/margin
layout_gravity：描述该view与父view间位置关系，即view的布局描述
gravity：描述该view与其内容的位置关系，比如EditText/Button中的文字位置
需要注意的是，垂直布局中，描述左右方向的位置描述生效，垂直方向比如top、bottom不生效；水平布局类推。

padding：描述该view与其内容的距离关系
margin：描述该view与邻居view的距离关系
spacing：gallery中图片间隔
android:verticalSpacing/android:horizontalSpacing：gridview中图片上下、左右之间的间隔

####代码混淆
可以用sdk/tools下的混淆工具：proguard
更方便的是在工程文件project.properties中打开这行的注释，使用默认的配置文件：
proguard.config=${sdk.dir}/tools/proguard/proguard-android.txt:proguard-project.txt

####ZIP对齐对apk优化
签名之后进行此优化

```
zipalign -v 4 src.apk dest.apk // -v像是详细输出，4字节对齐
zipalgin -c -v 4 dest.apk   //-c检查对齐
```

####自动化测试工具：Monkey
monkey -p pkg_name -v event_times

####性能分析工具：traceview

```
Debug.startMethodTracing("activity_trace");
test1();
test2();
Debug.stoipMethodTracing();
```

会在sd卡根目录生产一个activity_trace.trace文件，然后traceview activity_trace.trace出现分析界面。

####内存消耗测试

```
long total = Runtime.getRuntime().totalMemory();//系统总内存
long free = Runtime.getRuntime().freeMemory(); //剩余内存
```

####ANR的处理思路
将耗资源的操作(比如下载文件，复杂计算)放到其他线程。

####Listview 的优化
1. 利用convertView，convertView相当于一个缓存，convertView会指向上下滑动list时候最近被遮挡的一个view。如果它不为空可直接拿这个view来使用，只需要重新更新相应的数据即可，而不需要重新new出一个view。
2. 定义一个ViewHolder类，存放需要findViewById查找的控件，然后使用setTag方法将ViewHolder对象与View绑定。在getView方法中convertView不为空的时候即可通过getTag方法重新得到view相应的控件，减少了消耗性能的findViewById操作的使用。
更多详情见：[Android ListView使用BaseAdapter与ListView的优化](http://www.open-open.com/lib/view/open1339485728006.html)

####组件所在进程
组件运行在哪个进程中可以在AndroidManifest中配置，其中Activity/Service/receiver/provider都有一个process属性来指定该组件运行所在进程。
默认情况下，Android为每个应用程序分配一个唯一的User Id。然而，如果有多个应用程序都将该属性设置为一个相同的值，那么它们将共享相同的Id。如果这些应用程序再被设置成运行在一个相同的进程，它们便可以彼此访问对方的数据

####内存回收
针对像bitmap这样占用空间比较大的对象，如下方式提醒系统及时回收是个好习惯。

```
if(bitmapObjec.isRecycled() == false){
    bitmapObject.recycle();
    system.gc();//提醒系统及时回收
}
```

####屏幕适配
1. 涉及 屏幕尺寸、屏幕密度、屏幕方向、屏幕分辨率、独立于屏幕密度的像素(dp/sp)
2. 限制屏幕尺寸：在AndroidManifestxml中通过<compatible-screens>或者<supports-screens>标签来限制屏幕尺寸。在GooglePlay或者其他App市场上来过滤。
为不同的屏幕尺寸提供不同的布局：屏幕尺寸分为4个等级：small(标准120dpi-240*320)/normal(标准160dpi-320*480)/large(标准240dpi-480*800)/xlarge(320dpi-)，相应的布局资源目录layout-small/layout-normal/layout-large/layout-xlarge
3. 为不同的屏幕密度提供不同分辨率的图像资源（区分界限模糊）：低、中、高密度屏幕对应drawable-ldpi(版本不同可能是drawable)/drawable-mdpi/drawable-hdpi
4. 每一种屏幕尺寸都有要求的最小屏幕长宽尺寸，这些最小长宽尺寸使用与屏幕密度无关的单位dp，4种泛化的屏幕尺寸对应的最小屏幕长宽尺寸如下：
xlarge: 960dp * 720 dp
large: 640dp * 480dp
normal: 470dp * 320dp
small: 426dp * 320dp

####有关数据库_id
Android对数据库表有一个约定，就是每张表都应该至少有_id这列。listview在适配cursor的时候，会默认的获取 _id 这列的值，如果建的表没有 _id这列的话，会报错。
如果不是用在listview中，创建表的时候可以不加 _id，但是查询的时候需要用as _id 。

####自定义ContentProvider的步骤

####有关耗电
耗电状态切换时间：
1. 尽可能的将数据放在一起，减少建立链接的次数
2. 使用数据缓存机制，一般来说，每2到5分钟读取一次数据，每次1到5M是最佳选择；避免下载重复数据
3. 使用GCM(Google Cloud Messaging for Android)推送App的更新通知，这样不需要单独建立网络连接进行更新

4. 设置核实的轮询频率，最好允许用户修改轮询的频率
5. 使用HttpURLConnection 的Cache
6. 使用wifi，大多数情况下，wifi对电量的消耗要小于移动网络
7. 使用更高的带宽，使数据尽快下载或上传完，缩短耗电高枫的时间。
ConnectivityManager.TYPE_WIFI/TYPE_MOBILE。

####硬件feature
android是默认设备带有触摸屏的，所以像在机顶盒、电视允许的系统中允许的话，就可以在AndroidManifest中配置不需要触摸屏：<uses-feature android:name="android.hardware.touchscreen" android:required="false">，运行时检测功能可用性可以用接口
getPackageManager().hasSystemFeature("android.hardware.telephony")

####framelayout使用场景
framelayout 不需要调整的时候使用，效率比较高

####scrollview
scrollview 把原来layout中LinearLayout之类的字段替换成ScrollView，再将其中的内容用LinearLayout包裹一下即可

res中资源名字不能用数字打头，而且不能大写

####intent.setDataAndType
intent.setdata settype互斥，可以看下代码在设置一个的时候将另外一个置空了，所以需要两个都设置的时候调用intent.setDataAndType

####intent-filter
AndroidManifests.xml中给一个Activity指定的intent-filter可以有多个，intent-filter中的action、category、data也可以有多个。intent中设置的action只能有一个。

####eclipse创建android项目时，预览layout.xml文件时提示： This version of the rendering library is more recent than your version of ADT plug-in. Please update ADT plug-in，导致无法正常预览布局文件。
问题根源：SDK版本过高，ADT版本过低。可以调节预览页面右上角的android versiont to use when rendering layouts in eclipse，选择较低版本的api。如果不想每次手动调节这个东西，一则按照网上攻略选择help—>install new software升级tools(反正我是没有升级成功，翻墙状态下都根本刷不出来)，二则直接删掉较新的sdk，只留下document即可(删掉后出现了appcompat_v7报错的情况，删之，随便新建个工程即可附带重新生成)。

####
关于新版ADT创建项目时出现appcompat_v7的问题_百度经验
点击菜单栏"Project"，选择"Clean",然后点击"OK"，Clean完成后，你会发现appcompat_v7包出错。test包出现红色警告的问题已经解决了，只是test包还有黄色警告，那只是因为test的Java文件中import的类没有被使用，所以完全不用管它。
如果还不管用，那么右键appcompat_v7包，选择Properties，点"Android",如果Library栏下的"Is Library"方框没勾选，则勾选上，点击"OK"，然后再照以上步骤执行"Clean"操作即可。
既然appcompat_v7包是一个能让2.1以上全使用上4.0版本的界面的支持库，那么如图所示，我们建项目时直接把最小SDK选在Android4.0以上不就不需要这个支持库了吗？结果证明我们的想法是对的。

###用默认配置生成的工程会报错：R cannot be resolved to a variable
查看是没有R.java生成的，将appcompat_v7工程打开即可。将此工程关闭后采用默认配置

生成的工程会立即出现报错。

####broadcast Reciever机制运行效率是比较低的，所以不能用于发送高频率或者发数据的场景。

####使用bindService时候需要在service类中实现继承Binder的内部类，然后在onBind方法中返回该内部类的对象。否则在绑定service的时候没有onServiceConnected回调。

####一个service只会有一个实例，多次绑定不会调用多次onCreate。service的整个生命周期有系统控制，所以不能通过new的方式来获取service的实例，需要通过bindService的形式获取。在一个activity中调用startService创建了服务，在该activity退出的时候服务不会被销毁；调用bindService创建的服务，在activity退出的时候服务会被销毁。

####startActivity方式打开音频、视频、图片、txt、word、网址、电话、短信、邮箱等等各种文件或组件

实例：

```
Uri uri = Uri.fromFile(new File("/mnt/sdcard/ice.avi"));
intent.setDataAndType (uri, "video/*");
this.startActivity(intent);
```

####Android推荐使用SparseArray代替HashMap，性能方面会有改善。

####LayoutInflater

LayoutInflater这个类的作用类似于findViewById(),

不同点：

LayoutInflater是用来找layout下xml布局文件的,而且它会实例化
findViewById()是找具体xml布局文件下的具体widget控件，比如：Button按钮

####Debug快捷键
F5 进入方法
F6 逐行追踪
F7 跳出方法
F8 下一个断电或是结束Debug

####Can't create handler inside thread that has not called Looper.prepare()
子线程中handler需要跟looper一起使用

####android 中不允许图片资源的文件名中出现大写字母，且不能以数字开头。
####res\xml\oneXmlFile.xml: Invalid file name: must contain only [a-z0-9_.]

####mac下快捷键
comman+shift+f 格式化对齐
command+Fn+F11 运行apk
command+shift+o导包操作，推荐用这个，不仅可以导入需要的包，还能删掉多余的包
command+alt+上/下 复制当前行到上面/下面
alt+上/下 移动当前行到上面/下面
command+alt+左/右 前一个/后一个编辑的页面
command+Fn+F11 切换模拟器的横竖屏(genymotion也支持)
ctrl+shit+c 注释/反注释
选中一个变量后，ctrl+k定位到下一个

####handler 负责发送消息，Looper负责接收handler发送的消息，并直接把消息回传给handler自己。

####(0202)am启动apk

am start -n [component] ,此处的component类似于代码里边的包名加类名，不过此处类名可以是完整的类名也可以是简写
am start -n net.sunniwell.app.swsettings/.SWSettingsActivity
am start -n
net.sunniwell.app.swsettings/net.sunniwell.app.swsettings.SWSettingsActivity

15. layout 文件不能有大写字母，只能是a-z, 0-9, or _.

14.有关不同类中方法重名
抛异常信息： java.lang.ClassCastException: android.view.ViewGroup$LayoutParams cannot be cast to android.widget.Gallery$LayoutParams
像imgview.setLayoutParams(new Gallery.LayoutParams(200, 150));
与 iv.setLayoutParams(new ImageSwitcher.LayoutParams(ImageSwitcher.LayoutParams.WRAP_CONTENT,ImageSwitcher.LayoutParams.WRAP_CONTENT));
根据报错找到LayoutParams使用位置，加上限定类。默认是属于ViewGroup

像button4.setOnClickListener(new View.OnClickListener() {
与builder.setPositiveButton("确定", new DialogInterface.OnClickListener() {

13. testGallery
attrs.xml 内容不能补全，怎么写出来的?

12. 有关widget，TabHost/TabWidget/FrameLayout需要引用android中id中相应名称tabhost/tabs/tabcontent，要不然找不到tb或者tb.setup 抛空指针异常。
例如tabhost：不能用android:id="@+id/tabhost"，需要android:id="@android:id/tabhost"
相应的Tabhost tb = (TabHost) findViewById(android.R.id.tabhost);

11. imageButton 点击时候会有整个方块颜色闪动，可以将背景设置为透明来解决此问题。

10. 有关adapter创建时候制定adapter的选择模式是android自带layout中的一系列值，需要在R前加android限定
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_single_choice, theItem);

9. 有关快捷键
alt+s+f 文件格式化对齐
ctrl+1 弹出报错信息，其中比较好用的是强制类型转换Add cast
ctrl+/ 添加注释及取消注释
ctrl+e 打开的文件列表，切换文件
ctrl+F11 运行apk
ctrl+o 列出当前类中的方法列表
ctrl+m 放大缩小当前窗口
ctrl+PageDown/PageUp 切换选项卡，即切换不同文件
crtl+shift+t 查找当前工程中的方法
crtl+shift+r 查找当前工程中的文件
ctrl+shift+o 导入包

8. 像RadioGroup/checkbox这样有相同名字listener方法的组件，在import方法的时候会冲突，这个时候在new listenser接口的时候在前面加上包名限定，如：
sex.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener()
{

7. @+id 新建一个资源id到R.java中，@id 引用一个资源id。如果引用一个在当前代码之后定义的资源的id，可以先在此新建。后续再@+id也没有影响。

6. 创建工程时候指定的包名体现在androidManifest.xml中，在src中新建包名需要一致，否则发现AndroidManifest及gen下包名均是建工程时候起的包名。可以在AndroidManifest中更改，保存后会在gen下更新，如果出现多个包共存可以project--clean一下。

5. 不能在内部注释

4. 填充接口中参数，极有可能包含在某个类中，具体哪个类，可以将鼠标停留在接口上然后从如下提示信息中查看：

3. res-drawable 中图片名字不能数字打头

2. Didn't find class "*****Activity" on path: /data/app/*******.apk
--更改包名后可以看到 *.java中package 引用报名自动改变了，下面引用的R路径没有改变，另外AndroidManifest.xml 中package 包名没有改变。这都需要手动改正过来。

1. 运行时候弹框Your project contains error(s)，please fix them before running your application.
具体报错：A resource exists with a different case
--如下，报名包含了跟目录名相同的字段：BitmapShader1，修改包名后ok